

Surface Parametrization of Nonsimply Connected Planar Bézier Regions

Orest Shardt

*Department of Chemical and Materials Engineering, University of Alberta, Edmonton,
Alberta T6G 2V4, Canada*

John C. Bowman*

*Department of Mathematical and Statistical Sciences, University of Alberta, Edmonton,
Alberta T6G 2G1, Canada*

Abstract

A technique is described for constructing three-dimensional vector graphics representations of planar regions bounded by cubic Bézier curves, such as smooth glyphs. It relies on a novel algorithm for compactly partitioning planar Bézier regions into nondegenerate Coons patches. New optimizations are also described for Bézier inside–outside tests and the computation of global bounds of directionally monotonic functions over a Bézier surface (such as its bounding box or optimal field-of-view angle). These algorithms underlie the three-dimensional illustration and typography features of the \TeX -aware vector graphics language ASYMPTOTE.

Keywords: curved triangulation, Bézier surfaces, nondegenerate Coons patches, nonsimply connected domains, inside–outside test, bounding box, field-of-view angle, directionally monotonic functions, vector graphics, PRC, 3D \TeX , ASYMPTOTE

2010 MSC: 65D17, 68U05, 68U15

*Corresponding author

URL: <http://www.math.ualberta.ca/~bowman> (John C. Bowman)

1. Introduction

Recent methods for lifting smooth two-dimensional (2D) font data into three dimensions (3D) have focused on rendering algorithms for the Graphics Processing Unit (GPU) [15]. However, scientific visualization often requires 3D vector graphics descriptions of surfaces constructed from smooth font data. For example, while current CAD formats, such as the PDF-embeddable *Product Representation Compact* (PRC, *précis* in French) [2] format, allow one to embed text annotations, they do not allow text to be manipulated as a 3D entity. Moreover, annotations can only handle simple text; they are not suitable for publication-quality mathematical typesetting.

In this work, we present a method for representing arbitrary planar regions, including text, as 3D surfaces. A significant advantage of this representation is consistency: text can then be rendered like any other 3D object. This gives one complete control over the typesetting process, such as kerning details, and the ability to manipulate text arbitrarily (e.g. by transformation or extrusion) in a compact resolution-independent vector form. In contrast, rendering and mesh-generation approaches destroy the smoothness of the original 2D font data.

In focusing on the generation of 3D surfaces from 2D planar data, the emphasis of this work is not on 3D rendering but rather on the underlying procedures for generating vector descriptions of 3D geometrical objects. Vector descriptions are particularly important for online publishing, where no assumption can be made *a priori* about the resolution that will be used to display an image. As explained in Section 2, we focus on surfaces based on polynomial parametrizations rather than nonuniform rational B-splines (NURBS) [9, 19]. In Section 3 we describe a method for splitting an arbitrary planar region bounded by one or more Bézier curves into nondegenerate Bézier patches. This algorithm relies on the optimized Bézier inside–outside test described in Section 4. The implementation of these algorithms in the vector graphics language ASYMPTOTE, along with the optimized 3D sizing algorithms presented in Section 5, is discussed in Section 6.

Using a compact vector format instead of a large number of polygons to represent manifolds has the advantage of reduced data representation (essential for the storage and transmission of 3D scenes) and the possibility, using relatively few control points, of exact or nearly exact geometrical descriptions of mathematical surfaces. For example, in Appendix A we show that a sphere can be represented to 0.05% accuracy with just eight cubic

38 Bézier surface patches.

39 2. Bézier vs. NURBS Parametrizations

The atomic graphical objects in PostScript and PDF, Bézier curves and surfaces, are composed of piecewise cubic polynomial *segments* and tensor product *patches*, respectively. A segment $\gamma(t) = \sum_{i=0}^3 B_i(t)\mathbf{P}_i$ has four control points \mathbf{P}_i , whereas a surface patch is defined by sixteen control points \mathbf{P}_{ij} :

$$\sigma(u, v) = (x(u, v), y(u, v)) = \sum_{i,j=0}^3 B_i(u)B_j(v)\mathbf{P}_{ij}.$$

40 Here $B_i(u) = \binom{3}{i}u^i(1-u)^{3-i}$ is the i th cubic Bernstein polynomial. Just as a
 41 Bézier curve passes through its two end control points, a Bézier surface nec-
 42 essarily passes through its four corner control points. These special control
 43 points are called *nodes*. It is convenient to define the *convex hull* of a cubic
 44 Bézier segment or patch to be the convex hull (minimal enclosing polygon
 45 or polyhedron) of its control points. A *straight* segment is one in which the
 46 control points are colinear and the derivative of the Bézier parametrization
 47 is never zero (i.e. the control points are arranged in the same order as their
 48 indices).

49 It is often desirable to project a 3D scene to a 2D vector graphics for-
 50 mat understood by a web browser or high-end printer. Although NURBS
 51 are popular in computer-aided design [9] because of the additional degrees
 52 of freedom introduced by weights and general knot vectors, these benefits
 53 are tempered by both the lack of support for NURBS in popular 2D vector
 54 graphics formats (PostScript, PDF, SVG, EMF) and the algorithmic simpli-
 55 fications afforded by specializing to a Bézier parametrization. Bézier curves
 56 are also commonly used to describe glyph outlines. We therefore restrict
 57 our attention to (polynomial) Bézier curves and surfaces (even though both
 58 ASYMPTOTE and the 3D PRC format support NURBS).

59 Unlike their Bézier counterparts, NURBS are invariant under perspective
 60 projection. This is only an issue if projection is done before the rendering
 61 stage, as is necessary when a 2D vector representation of a curve or surface
 62 is constructed solely from the 2D projection of its control points. It is there-
 63 fore somewhat ironic that NURBS are much less widely implemented in 2D
 64 vector graphics formats than in 3D. In 3D vector graphics applications, pro-
 65 jection to 2D is always deferred until rendering time, so that the invariance

66 of NURBS under nonaffine projection is irrelevant. While NURBS provide
 67 exact parametrizations of familiar conic sections and quadric surfaces, non-
 68 trivial manifolds still need to be approximated as piecewise unions of under-
 69 lying exact primitives. We feel that the implementational simplicity of basic
 70 Bézier operations (computing subcurves and subsurfaces, points of tangency,
 71 normal vectors, bounding boxes, intersection points, arc lengths, and arc
 72 times) offsets for many practical applications the lower dimensionality of the
 73 Bézier subspace.

74 3. Partitioning Curved 2D Regions

75 In 3D graphics, text is often displayed with bit-mapped images, textures,
 76 or polygonal mesh approximations to smooth font character curves. To allow
 77 viewing of smooth text at arbitrary magnifications and locations, a nonpolyg-
 78 onal surface that preserves the curvature of the boundary curves is required.
 79 While it is easy to fill the outline of a smooth character in 2D, filling a 3D
 80 planar surface requires more sophisticated methods. One approach involves
 81 using surface filling algorithms for execution on GPUs [15]. When a vec-
 82 tor, rather than a rendered, image is desired, a preferable alternative is to
 83 represent the text as a parametrized surface.

84 Methods based on common surface primitives in 3D modelling and ren-
 85 dering can be used to describe planar regions. One method trims the domain
 86 of a planar surface to the desired shape [17]. While that approach is feasible,
 87 given adequate software support for trimming, this work describes a differ-
 88 ent approach, where each symbol is represented as a set of planar Bézier
 89 patches. We call this procedure *bezulation* since it involves a process similar
 90 to the triangulation of a polygon but uses cubic Bézier patches instead of
 91 triangles. To generate a surface representing the region bounded by a set of
 92 *simple closed* Bézier curves (intersecting only at the end points), algorithms
 93 were developed for (i) expressing a simply connected 2D region as a union of
 94 Bézier patches and (ii) breaking up a nonsimply connected region into sim-
 95 ply connected regions. (Selfintersecting curves can be handled by splitting at
 96 the intersection points.) These algorithms allow one to express text surfaces
 97 conveniently as Bézier patches.

98 Bezulation of a simply connected planar region involves breaking the re-
 99 gion up into patches bounded by closed Bézier curves with four or fewer
 100 segments. This is performed by the routine `bezulate` (cf. Algorithm 1) us-
 101 ing an adaptation of a naïve triangulation algorithm, modified to handle

102 curved edges, as illustrated in Figure 1.

```

103 Input: simple closed curve C
Output: array of closed curves A
while C.segments > 4 do
    found  $\leftarrow$  false;
    for  $n = 3$  to 2 do
        for  $i = 0$  to C.segments-1 do
            L  $\leftarrow$  line segment between nodes  $i$  and  $i + n$  of C;
            if countIntersections(C,L) = 2 and midpoint of L is
               inside C then
                p  $\leftarrow$  subpath of C from node  $i$  to  $i + n$ ;
                q  $\leftarrow$  subpath of C from node  $i + n$  to  $i +$  C.segments;
                A.push(p+L);
                C  $\leftarrow$  L + q;
                found  $\leftarrow$  true;
                break;
            end
        end
        if found then
            | break;
        end
    end
    if not found then
        | refine C by inserting an additional node at the parametric
        | midpoint of each segment;
    end
end

```

Algorithm 1: bezulate partitions a simply connected region.

104 A line segment lies within a closed curve when it intersects the curve
105 only at its endpoints and its midpoint lies strictly inside the curve. If after
106 checking all connecting line segments between nodes separated by $n = 3$ or
107 $n = 2$ segments, none of them lie entirely inside the shape, the original curve
108 is refined by dividing each segment of the curve at its parametric midpoint.
109 The bezulation process then continues with the refined curve. This algorithm
110 can be modified to subdivide more optimally, for example, to avoid elongated
111 patches that sometimes lead to rendering problems.

112 If the region is convex, Algorithm 1 is easily seen to terminate: all con-

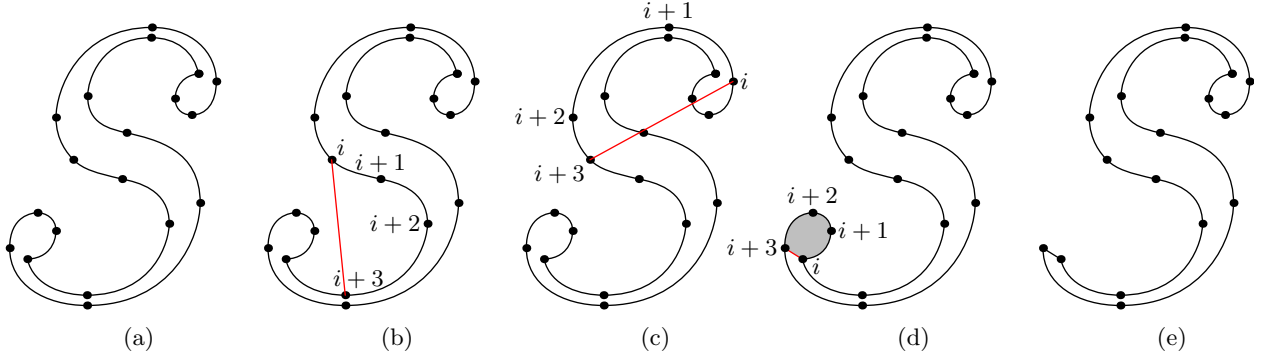


Figure 1: The **bezulate** algorithm. Starting with the original curve (a), several possible connecting line segments (shown in red) between nodes separated by $n = 3$ or $n = 2$ segments are tested. Connecting line segments are rejected if they do not lie entirely inside the original curve. This occurs when the midpoint is not inside the curve (b) or when the connecting line segment intersects the curve more than twice (c). If a connecting line segment passes both tests, the shaded section is separated (d) and the algorithm continues with the remaining curve (e).

113 necting line segments are admissible, and each patch removal decreases the
 114 number of points in the curve. Moreover, from the point of view of Algo-
 115 rithm 1, upon sufficient subdivision a non-convex region eventually becomes
 116 indistinguishable from a polygon, in which case the algorithm reduces to a
 117 straightforward polygonal triangulation.

118 3.1. Nonsimply Connected Regions

119 Since the **bezulate** algorithm requires simply connected regions, nonsim-
 120 ply connected regions must be handled specially. The “holes” in a nonsimply
 121 connected domain can be removed by partitioning the domain into a set of
 122 simply connected regions, each of which can then be bezulated.

123 For convenience we define a *top-level curve* to be a curve that is not
 124 contained inside any other curve and an *outer (inner) curve* to be the outer
 125 (inner) boundary of a filled region. With these definitions, the glyph “%”
 126 has two inner curves and two top-level curves that are also outer curves.

127 The algorithm proceeds as follows. First, to determine the topology of
 128 the region, the curves are sorted according to their relative insidedness, as
 129 determined by the nonzero winding number rule. Since the curves are as-
 130 summed to be simple, any point on an inner curve can be used to test whether
 131 that curve is inside another curve. The result of this sorting is a collection

132 of top-level curves grouped with the curves they surround. Each of these
 133 groups is treated independently.

134 Figure 3 illustrates the **partition** routine (cf. Algorithm 2). Each group
 135 is examined recursively to identify regions bounded by inner and outer curves.
 136 First, the inner curves in the group are sorted topologically to find the inner
 137 curves that are top-level curves with respect to the other inner curves. The
 138 inner curves that are not top-level curves are processed with a recursive call to
 139 **partition**. The nonsimply connected region between the outer (top-level)
 140 curve and the inner (top-level) curves is now split into simply connected
 141 regions. This is illustrated in Figure 2. The intersections of the inner and
 142 outer curves with a line segment from a point on an inner curve to a point
 143 on the outer curve are found (either *via* subdivision or a numerically robust
 144 cubic root solver). Consecutive intersections of this line segment, at points
 145 A and B , on the inner and outer curves, respectively, are selected. Let t_B
 146 be the value of the parameter used to parameterize the outer curve at B .
 147 Starting with $\Delta = 1$, Δ is halved until the line segment \overline{AC} , where C is
 148 the point on the outer curve at $t_B + \Delta$, does not intersect the outer curve
 149 more than once, does not intersect any inner curve (other than once at A),
 150 and the region bounded by \overline{AB} , \overline{AC} , and \widehat{BC} does not contain any inner
 151 curves. Once Δ and the point C have been found, the outer curve, less
 152 the segment between B and C , is merged with \overline{BA} , followed by the inner
 153 curve and then \overline{AC} . The region bounded by \overline{AB} , \overline{AC} , and \widehat{BC} is a simply
 154 connected region. Additional simply connected regions are found when the
 155 outer curve is merged with the other inner curves. Once the merging with
 156 all inner curves has been completed, the outer curve becomes the boundary
 157 of the final simply connected region.

158 The recursive algorithm for partitioning nonsimply connected regions into
 159 simply connected regions is summarized below. The function **sort** returns
 160 groups of top-level curves and the curves they contain. However, it is not
 161 recursive; the inner curves are not sorted. The function **merge** returns the
 162 simply connected regions formed from the single outer curve and multiple

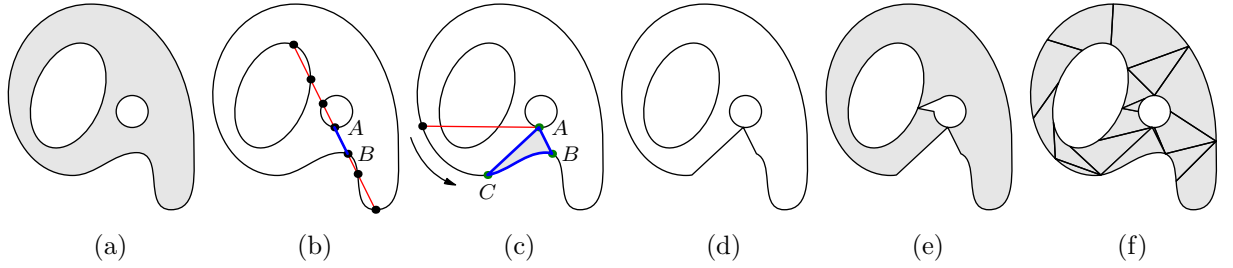


Figure 2: Splitting of non-simply connected regions into simply connected regions. Starting with a non-simply connected region (a), the intersections between each curve and an arbitrary line segment from a point on an inner curve to the outer curve are found (b). Consecutive intersections of this line segment, at points A and B , on the inner and outer curves, respectively, identify a convenient location for extracting a region. One searches along the outer curve for a point C such that the line segment AC intersects the outer curve no more than once, intersects an inner curve only at A , and determines a region ABC between the inner and outer curves that does not contain an inner curve. Once such a region is found (c), it is extracted (d). This extraction merges the inner curve with the outer curve. The process is repeated until all inner curves have been merged with the outer curve, leaving a simply connected region (e) that can be split into Bézier surface patches. The resulting patches and extracted regions are shaded in (f).

163 inner curves that are supplied to it.

Input: array of simple closed curves C
Output: array of closed curves A
foreach *group of nested curves* G **in** `sort(C)` **do**
 $\text{innerGroups} \leftarrow \text{sort}(G.\text{innerCurves});$
 foreach *group of nested curves* H **in** innerGroups **do**
 $A.\text{push}(\text{partition}(H.\text{innerCurves}));$
 end
 $A.\text{push}(\text{merge}(G.\text{toplevel}, \text{top-level curves of all groups in } \text{innerGroups}));$
end
return $A;$

164

Algorithm 2: `partition` splits nonsimply connected regions into simply connected regions. The pseudo-code functions `sort` and `merge` are described in the text.

The routines `bezulate` and `partition` were used to typeset the T_EX

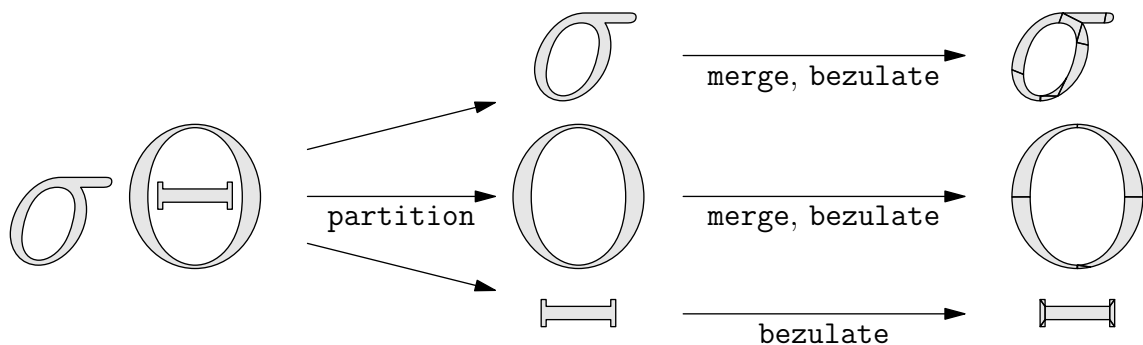


Figure 3: Illustration of the **partition** algorithm. The five curves that define the outlines of the Greek characters σ and Θ are passed in a single array to **partition**.

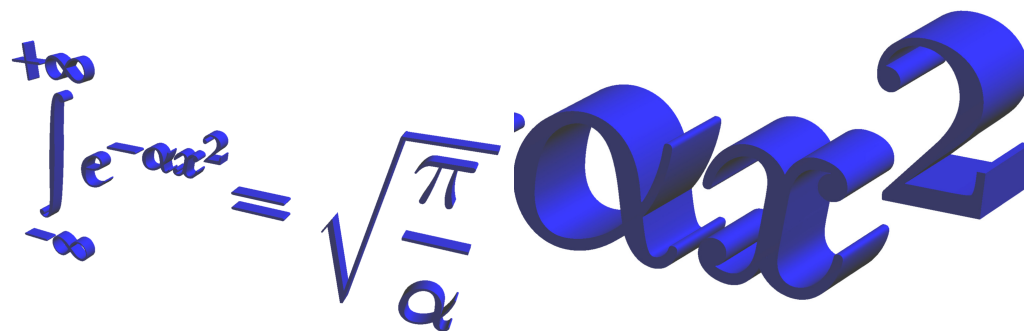


Figure 4: Application of the **bezulate** and **partition** algorithms to lift the Gaussian integral to three dimensions.

Figure 5: Zoomed view of Figure 4 generated from the same vector graphics data. The smooth boundaries of the characters emphasize the advantage of a 3D vector font description.

$$\int_{-\infty}^{+\infty} e^{-\alpha x^2} = \sqrt{\frac{\pi}{\alpha}}$$

Figure 6: Subpatch boundaries for Figure 4 as determined by the **bezulate** and **partition** algorithms.

equation

$$\int_{-\infty}^{+\infty} e^{-\alpha x^2} = \sqrt{\frac{\pi}{\alpha}}$$

165 in the interactive 3D diagram shown in Figure 4 and magnified, to emphasize
166 the smooth font boundaries, in Figure 5. The computed subpatch boundaries
167 are indicated in Figure 6.

168 Figure A.12 in Appendix Appendix A illustrates how **bezulate** is used
169 in mathematical drawings to lift \TeX to three dimensions. Referring to
170 the interactive 3D PDF version of this article¹ one see that the labels in
171 Figure A.12 have been programmed to rotate interactively so that they always
172 face the camera; this feature, implemented with **Javascript**, is known as
173 *billboard interaction*.

174 Developing Bézier versions of more sophisticated triangulation algorithms
175 would be an interesting future research project. The rendering technique
176 of Ref. [15] could be modified to produce Bézier patches, but this would
177 produce more patches than **bezulate**. For example, the “e” shown in Fig. 3
178 of Ref. [15] corresponds to roughly twice as many (4-segment) patches as
179 the ten patches generated by **bezulate** for the “e” in Fig. 6. Since our
180 interest is in compact 3D vector representations, the objective of this work
181 is to minimize the number of generated patches. In contrast, in real-time
182 rendering, one aims to minimize the overall execution time.

183 3.2. Nondegenerate Planar Bézier Patches

184 The **bezulate** algorithm described previously decomposes regions bounded
185 by closed curves (according to the nonzero winding number rule) into subre-
186 gions bounded by closed curves with four or fewer segments. Further steps are
187 required to turn these subregions into nondegenerate Bézier patches. First, if
188 the interior angle between the incoming and outgoing tangent directions at a
189 node is greater than 180° , the boundary curve is split at this node by follow-
190 ing the interior angle bisector to the first intersection with the path. This is
191 done to guarantee that the patch normal vectors at the nodes all point in the
192 same direction. Next, curves with less than four segments are supplemented
193 with null segments (four identical control points) to bring their total number
194 of segments up to four. A closed curve with four segments defines the twelve

¹See <http://asymptote.sourceforge.net/articles/>.

195 boundary control points of a Bézier patch in the x - y plane. The remaining
 196 four interior control points $\{\mathbf{P}_{11}, \mathbf{P}_{12}, \mathbf{P}_{21}, \mathbf{P}_{22}\}$ are then chosen to satisfy the
 197 Coons interpolation [7, 10, 1]

$$\begin{aligned} \boldsymbol{\sigma}(u, v) = \sum_{i=0}^3 & [(1-v)B_i(u)\mathbf{P}_{i,0} + vB_i(u)\mathbf{P}_{i,3} + (1-u)B_i(v)\mathbf{P}_{0,i} + uB_i(v)\mathbf{P}_{3,i}] \\ & - (1-u)(1-v)\mathbf{P}_{0,0} - (1-u)v\mathbf{P}_{0,3} - u(1-v)\mathbf{P}_{3,0} - uv\mathbf{P}_{3,3}. \end{aligned}$$

The resulting mapping $\boldsymbol{\sigma}(u, v)$ need not be bijective [22, 24, 14], even if the corner control points form a convex quadrilateral (despite the fact that a Coons patch for a convex polygon is always nondegenerate). In terms of the 2D scalar cross product $\mathbf{p} \times \mathbf{q} = p_x q_y - p_y q_x$, the Coons patch is seen to be a diffeomorphism of the unit square $D = [0, 1] \times [0, 1]$ if and only if the Jacobian

$$J(u, v) = \frac{\partial(x, y)}{\partial(u, v)} = \nabla_u x \times \nabla_v y = \sum_{i,j,k,\ell=0}^3 B'_i(u)B_j(v)B_k(u)B'_\ell(v)\mathbf{P}_{ij} \times \mathbf{P}_{k\ell}$$

198 (the z component of the corresponding 3D normal vector) is sign-definite.
 199 Since $J(u, v)$ is a continuous function of its arguments, this means that J
 200 must not vanish anywhere on D . A sign reversal of the Jacobian can manifest
 201 itself as an outright overlap of the region bounded by the curve or as an
 202 internal multivalued wrinkle, as illustrated in Figure 7. Rendering problems,
 203 such as the black smudges visible in Figures 7(b) and (e), can occur where
 204 isolines collide.

Randrianarivony and Brunnett [22] (and later H. Lin *et al.* [14]) describe sufficient conditions for $J(u, v)$ to be nonzero throughout D . In the case of a cubic Bézier patch, the 36 quantities

$$T_{pq} = \sum_{i+k=p} \sum_{j+\ell=q} \mathbf{U}_{i,j} \times \mathbf{V}_{k,\ell} \binom{2}{i} \binom{3}{k} \binom{3}{j} \binom{2}{\ell} \quad p, q = 0, 1, \dots, 5,$$

205 where $\mathbf{U}_{i,j} = \mathbf{P}_{i+1,j} - \mathbf{P}_{i,j}$ and $\mathbf{V}_{i,j} = \mathbf{P}_{i,j+1} - \mathbf{P}_{i,j}$, are required to be of
 206 the same sign. This follows from the fact that $J(u, v) = \sum_{p,q=0}^5 T_{pq} u^p v^q (1-u)^{5-p} (1-v)^{5-q}$.
 207

208 Randrianarivony *et al.* show further that every degenerate Coons patch
 209 can be decomposed into a finite union of nondegenerate subpatches (some
 210 with reversed orientation). However, the adaptive subdivision algorithm they

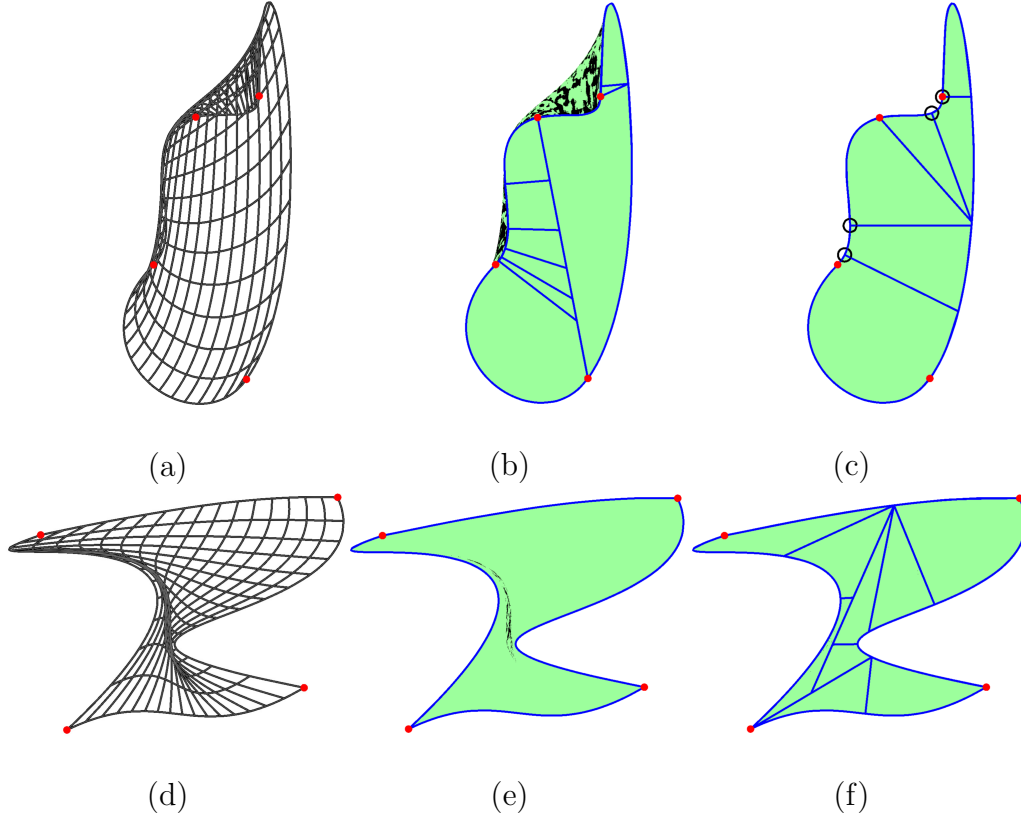


Figure 7: Degeneracy in a Coons patch. The dots indicate corner control points (nodes) and the open circles indicate the points of greatest degeneracy on the boundary, as determined by the quartic root solver: (a) overlapping isoline mesh; (b) overlapping patch; (c) nonoverlapping subpatches; (d) internally degenerate isoline mesh; (e) internally degenerate patch; (f) nondegenerate subpatches.

211 propose to exploit this fact does not prescribe an optimal boundary point
 212 at which to do the splitting. A better algorithm is based on the following
 213 elementary theorem, which provides a practical means of detecting Coons
 214 patches with degenerate boundaries.

Theorem 1 (Nondegenerate Boundary). *Consider a closed counter-clockwise oriented four-segment curve p in the x - y plane such that the interior angles formed by the incoming and outgoing tangent vectors at each node are less than or equal to 180° . Let $J(u, v)$ be the Jacobian of the corresponding Coons patch constructed from p , with control points \mathbf{P}_{ij} , and define the fifth-degree polynomial*

$$f(u) = \sum_{i,j=0}^3 B'_i(u)B_j(u)\mathbf{P}_{i,0} \times (\mathbf{P}_{j,1} - \mathbf{P}_{j,0}).$$

215 *If $f(u) \geq 0$ whenever $f'(u) = 0$ on $u \in (0, 1)$, then $J(u, 0) \geq 0$ on $[0, 1]$.*
 216 *Otherwise, the minimum value of $J(u, 0)$ occurs at a point where $f'(u) = 0$.*

Proof. First we note, since $B'_1(0) = -B'_0(0) = 3$ and $B'_2(0) = B'_3(0) = 0$, that $J(u, 0) = 3f(u)$ and

$$J(0, 0) = 3f(0) = 9(\mathbf{P}_{1,0} - \mathbf{P}_{0,0}) \times (\mathbf{P}_{0,1} - \mathbf{P}_{0,0}) \geq 0$$

217 since this is the cross product of the outgoing tangent vectors at $\mathbf{P}_{0,0}$. Like-
 218 wise, $J(1, 0) = 3f(1) \geq 0$. We know that the continuous function f must
 219 achieve its minimum value on $[0, 1]$ at some $u \in [0, 1]$. If f were negative
 220 somewhere in $(0, 1)$ we could conclude that $f(u) < 0$, so that $u \in (0, 1)$, and
 221 hence f would have an interior local minimum at u , with $f'(u) = 0$. But this
 222 is a contradiction, given that $f(u) \geq 0$ whenever $f'(u) = 0$. \square

The significance of Theorem 1 is that it affords a means of detecting a point u on the boundary where the Jacobian is most negative. This requires finding roots of the quartic polynomial

$$f'(u) = [B''_i(u)B_j(u) + B'_i(u)B'_j(u)]\mathbf{P}_{i,0} \times (\mathbf{P}_{j,1} - \mathbf{P}_{j,0}).$$

223 The coefficients of this quartic polynomial can be computed using the polyno-
 224 mials $M_{ij} = (B''_i B_j + B'_i B'_j)/3$ tabulated in Table 1. The method of Neumark
 225 [16], which relies on numerically robust cubic and quadratic root solvers, is
 226 then used to find algebraically all real roots of the quartic equation $f'(u) = 0$
 227 that lie in $(0, 1)$. The Jacobian is computed at each of these points; if it is

$$\begin{pmatrix} 5 - 20u + 30u^2 - 20u^3 + 5u^4 & -3 + 24u - 54u^2 + 48u^3 - 15u^4 & -6u + 27u^2 - 36u^3 + 15u^4 & -3u^2 + 8u^3 - 5u^4 \\ -7 + 36u - 66u^2 + 52u^3 - 15u^4 & 3 - 36u + 108u^2 - 120u^3 + 45u^4 & 6u - 45u^2 + 84u^3 - 45u^4 & 3u^2 - 16u^3 + 15u^4 \\ 2 - 18u + 45u^2 - 44u^3 + 15u^4 & 12u - 63u^2 + 96u^3 - 45u^4 & 18u^2 - 60u^3 + 45u^4 & 8u^3 - 15u^4 \\ 2u - 9u^2 + 12u^3 - 5u^4 & 9u^2 - 24u^3 + 15u^4 & 12u^3 - 15u^4 & 5u^4 \end{pmatrix}$$

Table 1: Coefficients of the polynomials $M_{ij} = (B''_i B_j + B'_i B'_j)/3$.

negative anywhere, the point where it is most negative is determined. The patch is then split along an interior line segment perpendicular to the tangent vector at this point. The next intersection point of the patch boundary with this line is used to split the patch into two pieces. Each of these pieces is then treated recursively (beginning with an additional call to **bezulate**, should the new boundary curve happen to have five segments).

If a patch possesses only internal degeneracies, like the one in Figure 7(d), the patch boundary is arbitrarily split into two closed curves, say along the perpendicular to the midpoint of some nonstraight side. The blue lines in Figures 7(b) and (f) illustrate such a midpoint splitting. The arguments of Randrianarivony *et al.* [22] establish that only a finite number of such subdivisions will be required to obtain a nondegenerate patch. Nondegenerate subpatches oriented in the direction opposite to the normal vector corresponding to the original oriented curve should be discarded to avoid rendering interference with correctly aligned overlying subpatches.

The blue lines in Figure 7(c) show that our quartic algorithm generates six subpatches, a substantial improvement over the nine subpatches produced by adaptive midpoint subdivision [22] in Figure 7(b). Figure 7(c) also emphasizes the ability of the quartic root algorithm to detect the optimal (most degenerate) points (circled) for splitting the boundary curve. As mentioned earlier, in both cases, it is possible that splitting can lead to curves with five segments. Such curves are split further by the **bezulate** algorithm so that any degeneracy of the resulting subpatches can be addressed.

Since an algebraic quartic root solver is an explicit algorithm, optimal subdivision of patches introduces minimal overhead compared to adaptive midpoint subdivision. In our implementation, the costs of adaptive midpoint subdivision for Figures 7(b) and Figure 7(f) were approximately the same. Using optimal subdivision in Figure 7(c) was 34% faster than adaptive

256 midpoint splitting, whereas there was only 2% additional overhead in check-
 257 ing for boundary degeneracy in Figure 7(f) (which possesses only internal
 258 degeneracy). Patches having only internal degeneracy arise relatively rarely
 259 in practice, but when they do, the subpatches obtained by adaptive midpoint
 260 subdivision also tend to exhibit internal degeneracy. Once internal degener-
 261 acy has been detected in a patch, we find that it is typically more efficient
 262 not to check its degenerate subpatches for boundary degeneracy (otherwise
 263 the overhead in checking for boundary degeneracy in Figure 7(f) would grow
 264 to 50%). Of course, since our interest is not in real-time rendering but in
 265 surface generation, the real advantage of optimal subdivision is that it can
 266 significantly reduce the number of generated patches (e.g. Figure 7(c) has
 267 one-third fewer patches than Figure 7(b)).

268 4. An Optimized Bézier Inside–Outside Test

269 Although PostScript has an `infill` function for testing whether a par-
 270 ticular point would be painted by the PostScript `fill` command, this is only
 271 an approximate digitized test corresponding to the resolution of the output
 272 device. Our `bezulate` routine requires a vector graphics algorithm, one that
 273 yields the winding number of an arbitrary closed piecewise Bézier curve about
 274 a given point.

275 A straightforward generalization of the standard ray-to-infinity method
 276 for computing winding numbers of a polygon about a point requires the so-
 277 lution of a cubic equation. As is well known, the latter problem can become
 278 numerically unstable as two or three roots begin to coalesce. While a con-
 279 ventional ray-curve (or ray-patch) intersection algorithm based on recursive
 280 subdivision [17] could be employed to count intersections by actually finding
 281 them, this typically entails excessive subdivision.

282 A more efficient but still robust subdivision method for computing the
 283 winding number of a closed Bézier curve arises from the topological obser-
 284 vation that if a point z lies outside the convex hull of a Bézier segment, the
 285 segment can be continuously deformed to a straight line segment between its
 286 endpoints, without changing its orientation relative to the point z . A given
 287 point will typically lie outside the convex hull of most segments of a Bézier
 288 curve. The orientation of these segments relative to the given point can be
 289 quickly and robustly determined, just as in the usual ray method for poly-
 290 gons, to determine the contribution, if any, to the winding number. For this

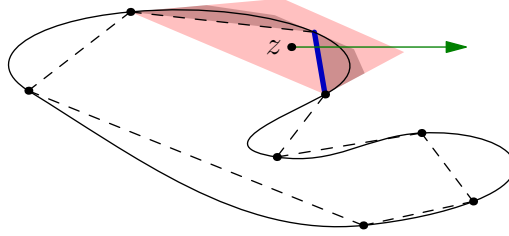


Figure 8: The **BézierWindingNumber** algorithm. Since z lies inside the convex hull of one Bézier segment, indicated by the light shaded region, that segment must be subdivided. On subdivision, z now lies outside the convex hulls of the subsegments, indicated by the dark shaded regions; these subsegments may be continuously deformed to straight line segments between their endpoints, without crossing z . The usual polygon inside–outside test may then be applied: the green ray establishes a winding number contribution of $+1$ due to the orientation of z with respect to the blue line.

291 purpose, Jonathan Shewchuk’s public-domain adaptive precision predicates
 292 for computational geometry [23] are highly recommended.

293 In the infrequent case where z lies on or inside the convex hull of a seg-
 294 ment, de Casteljau subdivision is used to split the Bézier segment about
 295 its parametric midpoint. Typically the convex hulls of the resulting sub-
 296 segments will overlap only at their common control point, so that z can lie
 297 strictly inside at most one of these hulls. This observation is responsible
 298 for the efficiency of the algorithm: one continues subdividing until the point
 299 is outside the convex hull of both segments or until machine precision is
 300 reached, as illustrated in Figure 8.

301 The orientation of segments whose convex hulls do not contain z can be
 302 handled by using the topological deformation property together with adap-
 303 tive precision predicates. Denoting by **straightContribution**(P, Q, z) the
 304 usual ray method for determining the winding number contribution of a line
 305 segment \overline{PQ} relative to a point z , the contribution from a Bézier segment S

306 can be computed as `curvedContribution(S,z)` (Algorithm 3).

Input: segment S , pair z
Output: winding number contribution of S about z
 $W \leftarrow 0$;
if z *lies within or on the convex hull of* S **then**
 foreach *subsegment* s *of* S **do**
 $W \leftarrow W + \text{curvedContribution}(s,z)$;
 end
else
 $W \leftarrow W + \text{straightContribution}(S.\text{beginpoint}, S.\text{endpoint}, z)$;
end
return W ;

Algorithm 3: `curvedContribution(S,z)` determines the winding number contribution from a Bézier segment S about z .

308 The winding number for a closed curve p about z may then be evaluated
309 with the algorithm `bézierWindingNumber(C,z)` (Algorithm 4).

Input: curve C , pair z
Output: winding number of C about z
 $W \leftarrow 0$;
foreach *segment* S *of* C **do**
 if S *is straight* **then**
 $W \leftarrow W + \text{straightContribution}(S.\text{beginpoint}, S.\text{endpoint}, z)$;
 else
 $W \leftarrow W + \text{curvedContribution}(S,z)$;
 end
end
return W ;

Algorithm 4: `bézierWindingNumber(C,z)` computes the winding number of a closed Bézier curve C about z .

311 A practical simplification of the above algorithm is the widely used op-
312 timization of testing whether a point is inside the 2D bounding box of the
313 control points rather than their convex hull. Since the convex hull of a Bézier
314 segment is contained within the bounding box of its control points, one can
315 replace “convex hull” by “control point bounding box” in the above algorithm
316 without modifying its correctness. One can easily check numerically that the
317 cost of the additional spurious subdivisions is well offset by the computational

318 savings in testing against the control point bounding box.

319 5. Global Bounds of Directionally Monotonic Functions

320 We now present efficient algorithms for computing global bounds of real-
 321 valued directionally monotonic functions $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ defined over a Bézier
 322 surface $\sigma(u, v)$. By *directionally monotonic* we mean that the restriction
 323 of f to each of the three Cartesian directions is a monotonic function; if f
 324 is differentiable this means that f has sign-semidefinite partial derivatives.
 325 These algorithms can be used to compute the 3D bounding box of a Bézier
 326 surface, the bounding box of its 2D projection, or the optimal field-of-view
 327 angle for sizing a 3D scene (cf. Fig. 9). The key observation is that the
 328 convex hull property of a Bézier patch holds independently in each direction
 329 and even under inversions like $z \rightarrow 1/z$.

330 A naïve approach to computing the bounding box of a Bézier patch re-
 331 quires subdivision whenever the 3D bounding boxes overlap in any of the
 332 three Cartesian directions. However, the number of required subdivisions
 333 can be greatly reduced by decoupling the three directions: in Algorithm 5,
 334 the problem is split into finding the maximum and minimum of the three
 335 *Cartesian axis projections* $f(x, y, z) = x$, $f(x, y, z) = y$, and $f(x, y, z) = z$
 336 evaluated over the patch. This requires a total of six applications of Al-
 337 gorithm 5. By convexity, the extrema of these special choices for f over a
 338 convex polyhedron \mathcal{C} occur at vertices of \mathcal{C} .

339 More general choices of directionally monotonic functions f are also of
 340 interest. For example, to determine the bounding box of the 2D perspective
 341 projection (based on similar triangles) of a surface, one can apply Algorithm 6
 342 in eye coordinates to the functions $f(x, y, z) = x/z$ and $f(x, y, z) = y/z$. This
 343 is useful for sizing a 3D object in terms of its 2D projection. For example,
 344 these functions were used to calculate the optimal field-of-view angle 13.4°
 345 for the Klein bottle shown in Figure 9.

346 For an arbitrary directionally monotonic function f , we note that

$$\sigma \subset \mathcal{C} \Rightarrow f(\sigma) \subset f(\mathcal{C}). \quad (1)$$

347 Our algorithms exploit Eq. (1) together with de Casteljau’s subdivision
 348 algorithm and the fact that a Bézier patch is confined to the convex hull of
 349 its control points. However, a patch is only guaranteed to intersect its convex
 350 hull at the four corner nodes.

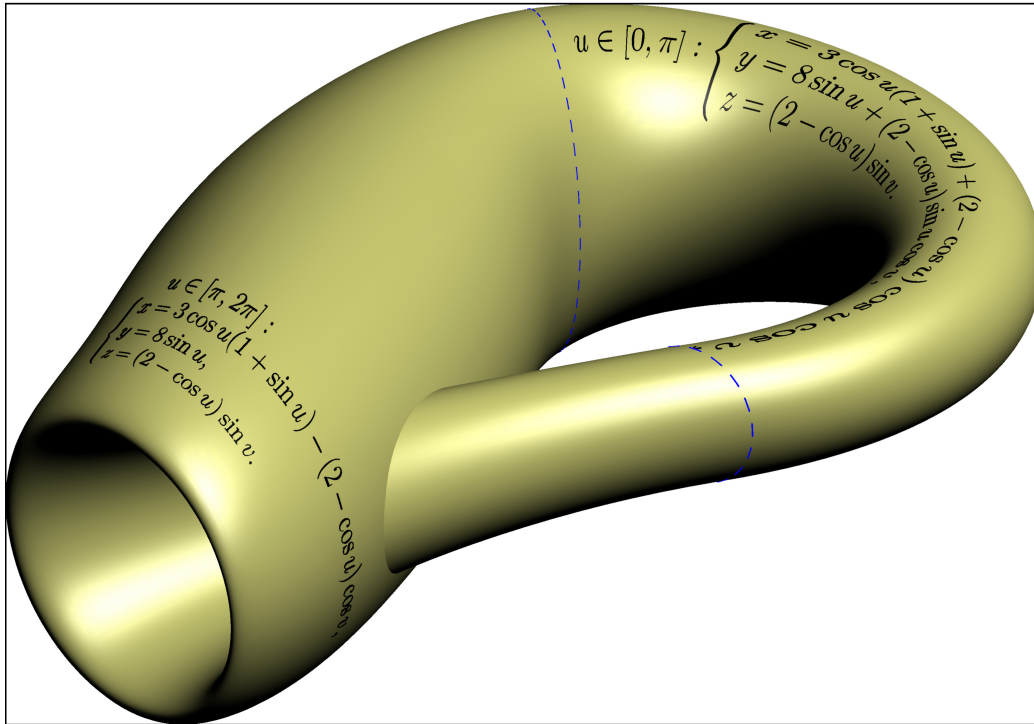


Figure 9: A Bézier approximation to a projection of a four-dimensional Klein bottle to three dimensions. The `FunctionMax` algorithm was used to determine the optimal field of view for this symmetric perspective projection of the scene from the camera location $(25.09, -30.33, 19.37)$ looking at $(-0.59, 0.69, -0.63)$. The extruded 3D \TeX equations embedded onto the surface provide a parametrization for the surface over the domain $u \times v \in [0, 2\pi] \times [0, 2\pi]$.

351 For the special case where f is a projection onto the Cartesian axes, the
 352 function **CartesianMax**($f, \mathbf{P}, f(\mathbf{P}_{00}), d$) given in Algorithm 5 computes the
 353 global maximum M of a Cartesian axis projection $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ over a Bézier
 354 patch \mathbf{P} to recursion depth d . Here, the value $f(\mathbf{P}_{00})$ provides a convenient
 355 starting value (lower bound) for M ; if the maximum of a surface consisting
 356 of several patches is desired, the value of M from previous patches is used
 357 to seed the calculation for the subsequent one. The algorithm exploits the
 358 fact that the extrema of each coordinate over the convex hull \mathcal{C} of \mathbf{P} occur
 359 at vertices of \mathcal{C} . First, one replaces M by the maximum of f evaluated at
 360 the four corner nodes and the previous value of M . If the maximum of the
 361 function evaluated at the remaining 12 control points is less than or equal
 362 to M , the subpatch can be discarded (by Eq. 1, noting that the maximum
 363 of $f(\mathcal{C})$ occurs at a control point and hence cannot exceed M). Otherwise,
 364 the patch is subdivided along the $u = v = 1/2$ isolines and the process is
 365 repeated using the new value of M . The method quickly converges to the
 366 global maximum of f over the entire patch.

Input: real function $f(\text{triple})$, patch \mathbf{P} , real M , integer **depth**
Output: real M
 $M \leftarrow \max(M, f(\mathbf{P}_{00}), f(\mathbf{P}_{03}), f(\mathbf{P}_{30}), f(\mathbf{P}_{33}));$
if **depth** = 0 **then**
 | **return** M ;
end
 $V \leftarrow \max(f(\mathbf{P}_{01}), f(\mathbf{P}_{02}), f(\mathbf{P}_{10}), f(\mathbf{P}_{11}), f(\mathbf{P}_{12}), f(\mathbf{P}_{13}),$
 $f(\mathbf{P}_{20}), f(\mathbf{P}_{21}), f(\mathbf{P}_{22}), f(\mathbf{P}_{23}), f(\mathbf{P}_{31}), f(\mathbf{P}_{32}));$
if $V \leq M$ **then**
 | **return** M ;
end
foreach *subpatch* S of \mathbf{P} **do**
 | $M \leftarrow \max(M, \text{FunctionMax}(f, S, M, \text{depth} - 1));$
end
return M ;

Algorithm 5: **CartesianMax**($f, \mathbf{P}, M, \text{depth}$) returns the maximum of M and the global bound of a Cartesian component f of a Bézier patch \mathbf{P} evaluated to recursion level **depth**.

367 For a general directionally monotonic function f (consider $f(x, y, z) = xy$

368 over $\mathcal{C} = \partial\{(x, y, 0) : 0 \leq x \leq 1, 0 \leq y \leq x\}$, the maximum of $f(\mathcal{C})$ need
 369 not occur at vertices of \mathcal{C} : one instead needs to examine the function value
 370 at the appropriate vertex of the bounding box of \mathcal{C} . For example, if f is a
 371 monotonic increasing function in each of the three Cartesian directions,

$$\mathcal{C} \subset \text{box}(\mathbf{a}, \mathbf{b}) \Rightarrow f(\mathcal{C}) \subset [f(\mathbf{a}), f(\mathbf{b})], \quad (2)$$

372 where $\text{box}(\mathbf{a}, \mathbf{b})$ denotes the 3D box with minimal and maximal vertices \mathbf{a}
 373 and \mathbf{b} , respectively.

374 The global maximum M of a directionally monotonic increasing function
 375 $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ over a Bézier patch \mathbf{P} can then be efficiently computed to
 376 recursion depth d by calling the function `FunctionMax`($f, \mathbf{P}, f(\mathbf{P}_{00}), d$) given
 377 in Algorithm 6. First, one replaces M by the maximum of f evaluated at
 378 the four corner nodes and the previous value of M . One then computes the
 379 vertex \mathbf{b} of the bounding box of the convex hull \mathcal{C} of \mathbf{P} . If the maximum of
 380 the function evaluated at \mathbf{b} is less than or equal to M , the subpatch can be
 381 discarded. Otherwise, the patch is subdivided along the $u = v = 1/2$ isolines
 382 and the process is repeated using the new value of M .

383 6. 3D Vector Typography

384 Donald Knuth’s T_EX system [13], the de-facto standard for typesetting
 385 mathematics, uses Bézier curves to represent 2D characters. T_EX provides
 386 a portable interface that yields consistent, publication quality typesetting of
 387 equations, using subtle spacing rules derived from centuries of professional
 388 mathematical typographical experience. However, while it is often desirable
 389 to illustrate abstract mathematical concepts in T_EX documents, no compati-
 390 ble descriptive standard for technical mathematical drawing has yet emerged.

391 The recently developed ASYMPTOTE language² aims to fill this gap by
 392 providing a portable T_EX-aware tool for producing 2D and 3D vector graph-
 393 ics [5]. In mathematical applications, it is important to typeset labels and
 394 equations with T_EX for overall consistency between the text and graphical el-
 395 ements of a document. In addition to providing access to the T_EX typesetting
 396 system in a 3D context, ASYMPTOTE also fills in a gap for nonmathematical

²available from <http://asymptote.sourceforge.net> under the GNU Lesser General Public License.

Input: real function $f(\text{triple})$, patch \mathbf{P} , real M , integer depth

Output: real M

$M \leftarrow \max(M, f(\mathbf{P}_{00}), f(\mathbf{P}_{03}), f(\mathbf{P}_{30}), f(\mathbf{P}_{33}));$

if $\text{depth} = 0$ **then**

return M ;

end

$x \leftarrow \max(\hat{\mathbf{x}} \cdot \mathbf{P}_{ij} : 0 \leq i, j \leq 3);$

$y \leftarrow \max(\hat{\mathbf{y}} \cdot \mathbf{P}_{ij} : 0 \leq i, j \leq 3);$

$z \leftarrow \max(\hat{\mathbf{z}} \cdot \mathbf{P}_{ij} : 0 \leq i, j \leq 3);$

if $f((x, y, z)) \leq M$ **then**

return M ;

end

foreach *subpatch* S of \mathbf{P} **do**

$M \leftarrow \max(M, \text{FunctionMax}(f, S, M, \text{depth} - 1));$

end

return M ;

Algorithm 6: $\text{FunctionMax}(f, \mathbf{P}, M, \text{depth})$ returns the maximum of M and the global bound of a real-valued directionally monotonic increasing function f over a Bézier patch \mathbf{P} evaluated to recursion level depth . Here $\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}$ are the Cartesian unit vectors.

397 applications. While open source 3D bit-mapped text fonts are widely avail-
398 able,³ resources currently available for scalable (vector) fonts appear to be
399 quite limited in three dimensions.

400 ASYMPTOTE was inspired by John Hobby’s METAFONT (a modified ver-
401 sion of METAFONT, the program that Knuth wrote to generate the T_EX
402 fonts), but is more powerful, has a cleaner syntax, and uses IEEE floating
403 point numerics. An important feature of ASYMPTOTE is its use of the simplex
404 linear programming method to solve overall size constraint inequalities be-
405 tween fixed-sized objects (labels, dots, and arrowheads) and scalable objects
406 (curves and surfaces). This means that the user does not have to scale man-
407 ually the various components of a figure by trial-and-error. The 3D versions
408 of ASYMPTOTE’s deferred drawing routines rely on the efficient algorithms
409 for computing the bounding box of a Bézier surface, along with the bounding
410 box of its 2D projection, described in Sec. 5. ASYMPTOTE natively generates
411 PostScript, PDF, SVG, and PRC [2] vector graphics output. The latter is a
412 highly compressed 3D format that is typically embedded within a PDF file
413 and viewed with the widely available ADOBE READER software.

414 The biggest obstacle that was encountered in generalizing ASYMPTOTE
415 to produce 3D interactive output was the fact that T_EX is fundamentally a
416 2D program. In this work, we have developed a technique for embedding
417 2D vector descriptions, like T_EX fonts, as 3D surfaces (2D vector graphics
418 representations of T_EX output can be extracted with a technique like that
419 described in Ref. [6]). While the general problem of filling an arbitrary 3D
420 closed curve is ill-posed, there is no ambiguity in the important special case
421 of filling a planar curve with a planar surface.

422 Since our procedure transforms text into Bézier patches, which are the
423 surface primitives used in ASYMPTOTE, all of the existing 3D ASYMPTOTE
424 algorithms can be used without modification. Together with the 3D gen-
425 eralization of the METAFONT curve operators described by [4, 5], these
426 algorithms comprise the 3D foundation for the T_EX-aware vector graphics
427 language ASYMPTOTE.

428 6.1. 3D Arrowheads

429 Arrows are frequently used in illustrations to draw attention to important
430 features. We designed curved 3D arrowheads that can be viewed from a

³For example, see <http://www.opengl.org/resources/features/fontsurvey/>.

431 wide range of angles. For example, the default 3D arrowhead was formed by
 432 bending the control points of a cone around the tip of a Bézier curve. Planar
 433 arrowheads derived from 2D arrowhead styles are also implemented; they are
 434 oriented by default on a plane perpendicular to the initial viewing direction.
 435 Examples of these arrows are displayed in Figures 10 and 11. The `bezulate`
 436 algorithm was used to construct the upper and lower faces of the filled (red)
 437 planar arrowhead in Fig. 11.

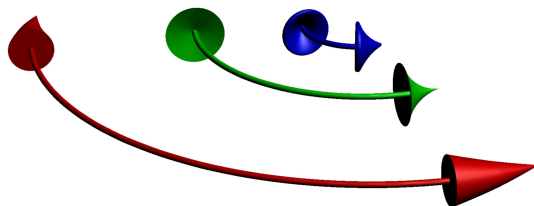


Figure 10: Three-dimensional revolved arrowheads in ASYMPTOTE.

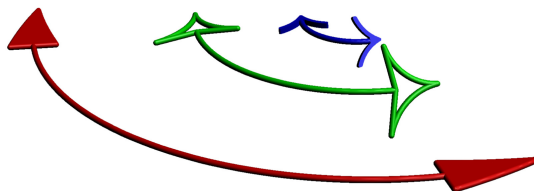


Figure 11: Planar arrowheads in ASYMPTOTE.

438 6.2. Double Deferred Drawing

439 Journal size constraints typically dictate the final width and height, in
 440 PostScript coordinates, of a 2D or projected 3D figure. However, it is often
 441 convenient for users to work in more physically meaningful coordinates. This
 442 requires *deferred drawing*: a graphical object cannot be drawn until the actual
 443 scaling of the user coordinates (in terms of PostScript coordinates) is known
 444 [5]. One therefore needs to queue a function that can draw the scaled object
 445 later, when this scaling is known. ASYMPTOTE's high-order functions provide
 446 a flexible mechanism that allows the user to specify either or both of the 3D
 447 model dimensions and the final projected 2D size. This requires two levels of
 448 deferred drawing, one that first sizes the 3D model and one that scales the
 449 resulting picture to fit the requested 2D size [6]. The 3D bounding box of

450 a Bézier surface, along with the bounding box of its 2D projection, can be
451 efficiently computed with the method described in Section 5.

452 6.3. *Efficient Rendering*

453 Efficient algorithms for determining the bounding box of a Bézier patch
454 also have an important application in rendering. Knowing the bounding box
455 of a Bézier patch allows one to determine, at a high level, whether it is in the
456 field of view: offscreen Bézier patches can be dropped before mesh generation
457 occurs [11]. This is particularly important for a spatially adaptive algorithm
458 as used in ASYMPTOTE’s OpenGL-based renderer, which resolves the patch
459 to one pixel precision at all zoom levels. Moreover, to avoid subdivision
460 cracks, renderers typically resolve visible surfaces to a uniform resolution.
461 It is therefore important that offscreen patches do not force an overly fine
462 mesh within the viewport. As a result of these optimizations, the native
463 ASYMPTOTE adaptive renderer is typically comparable in speed with the
464 fixed-mesh PRC renderer in ADOBE READER, even though the former yields
465 higher quality, true vector graphics output.

466 7. Conclusions

467 In this work we have developed methods that can be used to lift smooth
468 fonts, such as those produced by \TeX , into 3D. Treating 3D fonts as sur-
469 faces allows for arbitrary 3D text manipulation, as illustrated in Figures 5
470 and 9. The `bezulate` algorithm allows one to construct planar Bézier surface
471 patches by decomposing (possibly nonsimply connected) regions bounded by
472 simple closed curves into subregions bounded by closed curves with four or
473 fewer segments. The method relies on an optimized subdivision algorithm
474 for testing whether a point lies inside a closed Bézier curve, based on the
475 topological deformation of the curve to a polygon. We have also shown how
476 degenerate Coons patches can be efficiently detected and split into nondegen-
477 erate subpatches. This is required to avoid both patch overlap at the bound-
478 aries of the underlying curve and rendering artifacts (patchiness, smudges,
479 or wrinkles) due to normal reversal.

480 We have illustrated applications of these techniques in the open source
481 vector graphics programming language ASYMPTOTE, which we believe is the
482 first software to lift \TeX into 3D. This represents an important milestone for
483 publication-quality scientific graphing.

484 Appendix A. Bézier Approximation of a Sphere

485 As previously emphasized, although conic sections (quadrics) may be ac-
 486 curately represented by NURBS surfaces, the language of high-end printers,
 487 PostScript, supports only Bézier curves and surfaces. Although PostScript
 488 is only a 2D language, vector graphics projections of Bézier surfaces are
 489 nevertheless possible using tensor-product patch shading and hidden surface
 490 splitting along approximations to the visible surface horizon.

491 Here we illustrate that a sphere may be approximated to high graphical
 492 accuracy by a Bézier surface with only 8 patches, one for each octant, fol-
 493 lowing a procedure suggested in Ref. [18]. The patch describing an octant
 494 is degenerate at the pole: two of the nodes are placed there, with the other
 495 two placed along the equator, 90° apart in longitude.

496 Following Knuth, a unit quarter circle is approximated in ASYMPTOTE
 497 “with less than 0.06% error” [12], using the control points $\{(1, 0), (1, a), (a, 1), (0, 1)\}$,
 498 where $a = \frac{4}{3}(\sqrt{2} - 1)$. This value of a is determined by requiring that the
 499 third-order Bézier midpoint lie on the unit circle at $(1/\sqrt{2}, 1/\sqrt{2})$. (Other
 500 methods of approximating circular arcs by Bézier curves have been described
 501 in Refs. [3], [8], and [21].)

502 The above prescription immediately determines the three circular arcs
 503 describing the patch boundary for a unit spherical octant. Let us place
 504 \mathbf{P}_{00} at $(1, 0, 0)$, $\mathbf{P}_{03} = \mathbf{P}_{13} = \mathbf{P}_{23} = \mathbf{P}_{33}$ at $(0, 0, 1)$, and \mathbf{P}_{30} at $(0, 1, 0)$.
 505 The remaining control points $\{\mathbf{P}_{11}, \mathbf{P}_{12}, \mathbf{P}_{21}, \mathbf{P}_{22}\}$ are chosen to make the
 506 surface nearly spherical and the interface with adjacent octants smooth (have
 507 continuous first derivatives at the patch boundaries). The point \mathbf{P}_{11} is chosen
 508 (on the tangent plane at $x = 1$) to be the vector sum $\mathbf{P}_{10} + \mathbf{P}_{01} - \mathbf{P}_{00} =$
 509 $(1, a, 0) + (1, 0, a) - (1, 0, 0) = (1, a, a)$. We also require that the triangle
 510 in the x - y plane formed by the origin and the projections of \mathbf{P}_{12} onto the
 511 x - y plane and the x axis is similar to the corresponding triangle for \mathbf{P}_{11} .
 512 This implies that $\mathbf{P}_{12} = (a, a^2, 1)$. Similarly, we determine $\mathbf{P}_{22} = (a^2, a, 1)$
 513 and $\mathbf{P}_{21} = (a, 1, a)$. The final Bézier patch and resulting approximation to
 514 a unit sphere, with the control point mesh shown in blue, are illustrated
 515 in Figure A.12. We found numerically that the radius of this approximate
 516 sphere, generated with a 12×7 control point mesh, varies by less than 0.052%,
 517 well below the tolerance 0.1% to which Figure 8 of Ref. [20] was drawn using
 518 a much finer 22×13 control point mesh.

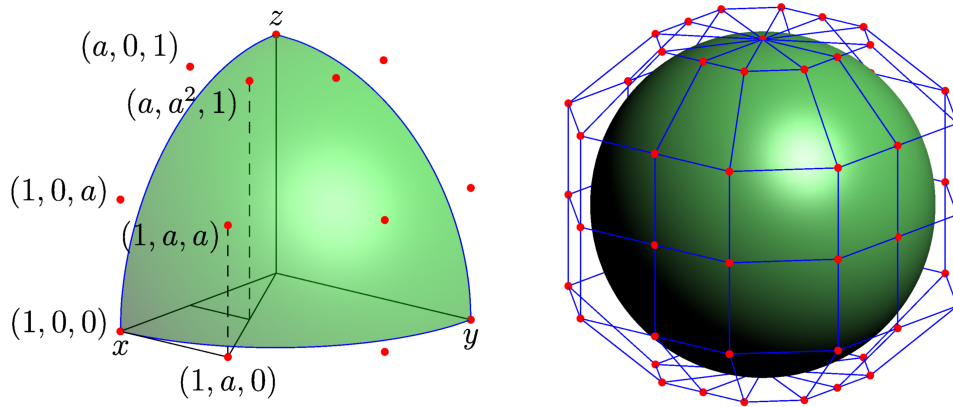


Figure A.12: Bézier approximation to a unit sphere. The red dots indicate control points.

Acknowledgments

We thank Philippe Ivaldi, Radoslav Marinov, Malcolm Roberts, Jens Schwaiger, and Olivier Guibé for discussions and assistance in implementing the algorithms described in this work. Special thanks goes to Andy Hammerlindl, who designed and implemented much of the underlying ASYMPTOTE language, and to Michail Vidiassov, who helped write the PRC driver. Financial support for this work was generously provided by the Natural Sciences and Engineering Research Council of Canada.

References

- [1] Adobe, July 2008. Document management—Portable Document Format—Part 1: PDF 1.7. ISO 32000-1:2008.
- [2] Adobe, 2008. PRC format specification.
http://livedocs.adobe.com/acrobat_sdk/9/Acrobat9_HTMLHelp/API_References/PRCReference/PRC_Format_Specification/.
- [3] Bézier, P., 1986. The Mathematical Basis of the UNISURF CAD System. Butterworths, London.
- [4] Bowman, J. C., 2007. The 3D Asymptote generalization of MetaPost Bézier interpolation. Proceedings in Applied Mathematics and Mechanics 7 (1), 2010021–2010022.

- 538 [5] Bowman, J. C., Hammerlindl, A., 2008. Asymptote: A vector graphics
539 language. TUGboat: The Communications of the T_EX Users Group
540 29 (2), 288–294.
- 541 [6] Bowman, J. C., Shardt, O., 2009. Asymptote: Lifting T_EX to three
542 dimensions. TUGboat: The Communications of the T_EX Users Group
543 30 (1), 58–63.
- 544 [7] Coons, S., 1964. Surfaces for computer aided design. Tech. rep., MIT,
545 Cambridge, Massachusetts, available as AD 663504 from the National
546 Technical Information service, Springfield, VA 22161.
- 547 [8] Fang, L., 1998. Circular arc approximation by quintic polynomial curves.
548 Computer Aided Geometric Design 15 (8), 843–861.
- 549 [9] Farin, G., 1992. From conics to nurbs: A tutorial and survey. IEEE
550 Computer Graphics and Applications 12 (5), 78–86.
- 551 [10] Farin, G., 2002. Curves and surfaces for CAGD: a practical guide. Mor-
552 gan Kaufmann, San Francisco, CA.
- 553 [11] Hasselgren, J., Munkberg, J., Akenine-Möller, T., 2009. Automatic pre-
554 tessellation culling. ACM Trans. Graph. 28 (2), 1–10.
- 555 [12] Knuth, D. E., 1986. The METAFONTbook. Addison-Wesley, Reading,
556 Massachusetts.
- 557 [13] Knuth, D. E., 1986. The T_EXbook. Addison-Wesley, Reading, Mas-
558 sachusetts.
- 559 [14] Lin, H., Tang, K., Joneja, A., Bao, H., 2007. Generating strictly non-self-
560 overlapping structured quadrilateral grids. Comput. Aided Des. 39 (9),
561 709–718.
- 562 [15] Loop, C., Blinn, J., 2005. Resolution independent curve rendering us-
563 ing programmable graphics hardware. In: SIGGRAPH '05: ACM SIG-
564 GRAPH 2005 Papers. ACM, New York, NY, USA, pp. 1000–1009.
- 565 [16] Neumark, S., 1965. Solution of cubic and quartic equations. Pergamon
566 Press, Oxford.

- 567 [17] Nishita, T., Sederberg, T. W., Kakimoto, M., 1990. Ray tracing trimmed
568 rational surface patches. SIGGRAPH Comput. Graph. 24 (4), 337–345.
- 569 [18] Ostby, E., 1988. Defining a sphere with bezier patches, [http://www.
570 gamedev.net/reference/articles/article407.asp](http://www.gamedev.net/reference/articles/article407.asp).
- 571 [19] Piegl, L., Tiller, W., 1997. The NURBS book. Springer, Berlin.
- 572 [20] Piegl, L., Tiller, W., 2003. Approximating surfaces of revolution by non-
573 rational B-splines. IEEE Computer Graphics and Applications 23 (3),
574 46–52.
- 575 [21] Piegl, L., Tiller, W., 2003. Circle approximation using integral B-splines.
576 Computer-Aided Design 35 (6), 601–607.
- 577 [22] Randrianarivony, M., Brunnett, G., May 2004. Necessary and suffi-
578 cient conditions for the regularity of planar Coons map. Tech. Rep.
579 SFB393/04-07, Technische Universität Chemnitz, Chemnitz, Germany.
- 580 [23] Shewchuk, J. R., Oct. 1997. Adaptive Precision Floating-Point Arith-
581 metic and Fast Robust Geometric Predicates. Discrete & Computational
582 Geometry 18 (3), 305–363.
- 583 [24] Wang, C. C. L., Tang, K., 2005. Non-self-overlapping Hermite interpola-
584 tion mapping: a practical solution for structured quadrilateral meshing.
585 Computer-Aided Design 37 (2), 271–283.